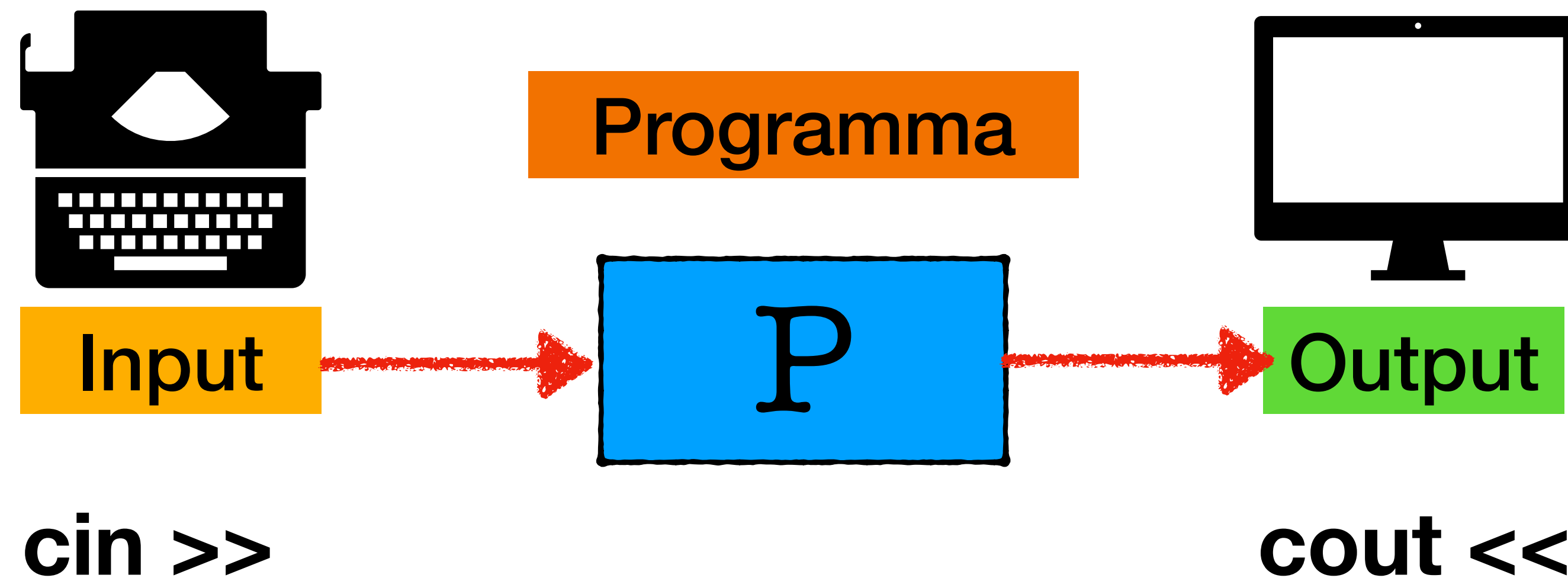


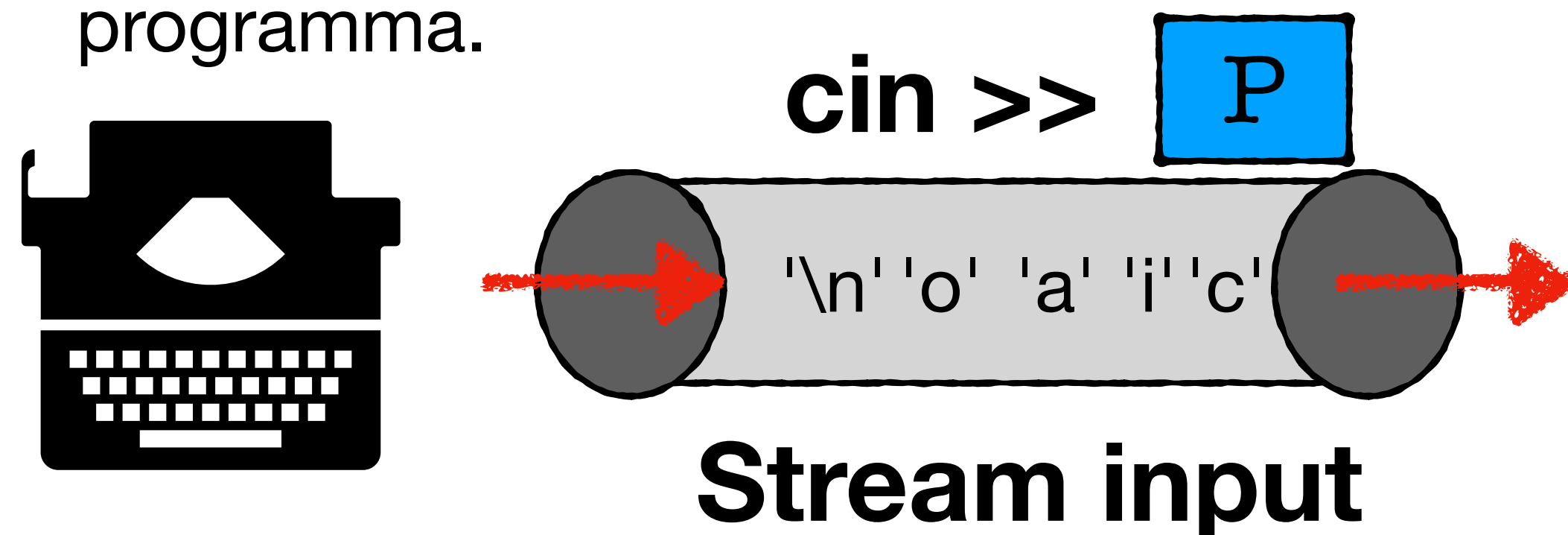
# Uso di files all'interno dei programmi

Stream, file stream

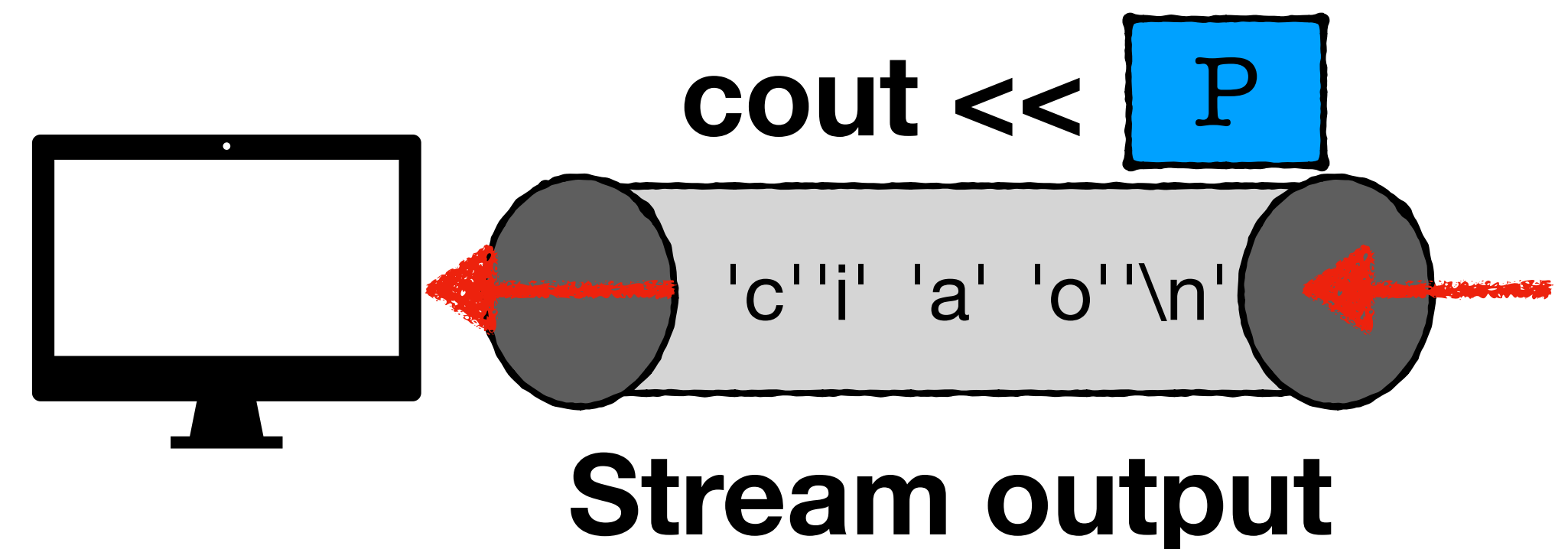
# Flussi di dati: cin, cout



- La tastiera è la SORGENTE di un flusso di informazioni, che viaggiano verso il programma.

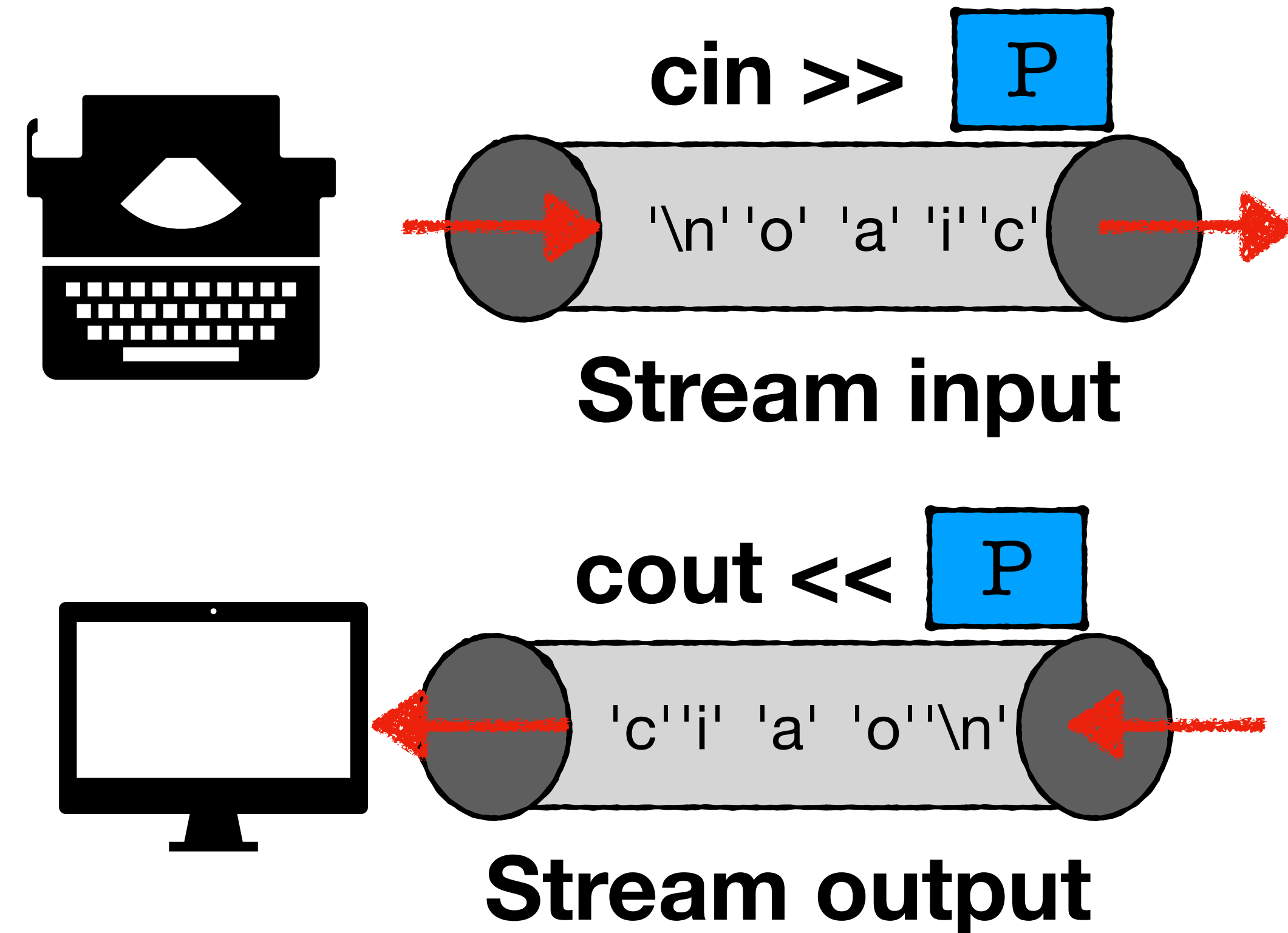


- Il monitor è la DESTINAZIONE di un flusso di informazioni, che originano dal programma.



# Flussi di dati: cin, cout

- Quando la libreria <iostream> viene usata, un **flusso di input** e un **flusso di output** vengono automaticamente associati alla tastiera e al video, rispettivamente.
- Gli stream sono entità su cui viaggiano e capaci di gestire caratteri (char).
- Gli operatori di estrazione (>>, <<) sono operatori "intelligenti" capaci di creare e/o accorpare sequenze di caratteri (stringhe) e di interpretarle in modo corretto. Ricordiamo che nel linguaggio tutto ha un tipo
- I caratteri spazio, tabulazione, a-capo, vengono trattati in modo uniforme come separatori tra pezzi (token) di informazione.



# Files

# Files

## Definizione:

- Un file è una sequenza finita di **caratteri**.
- Registrata nel disco fisso del nostro calcolatore.
- Identificata da un nome. In Linux: percorso completo dalla root al nome proprio del file incluso

/home/tama/Desktop/dati.dat

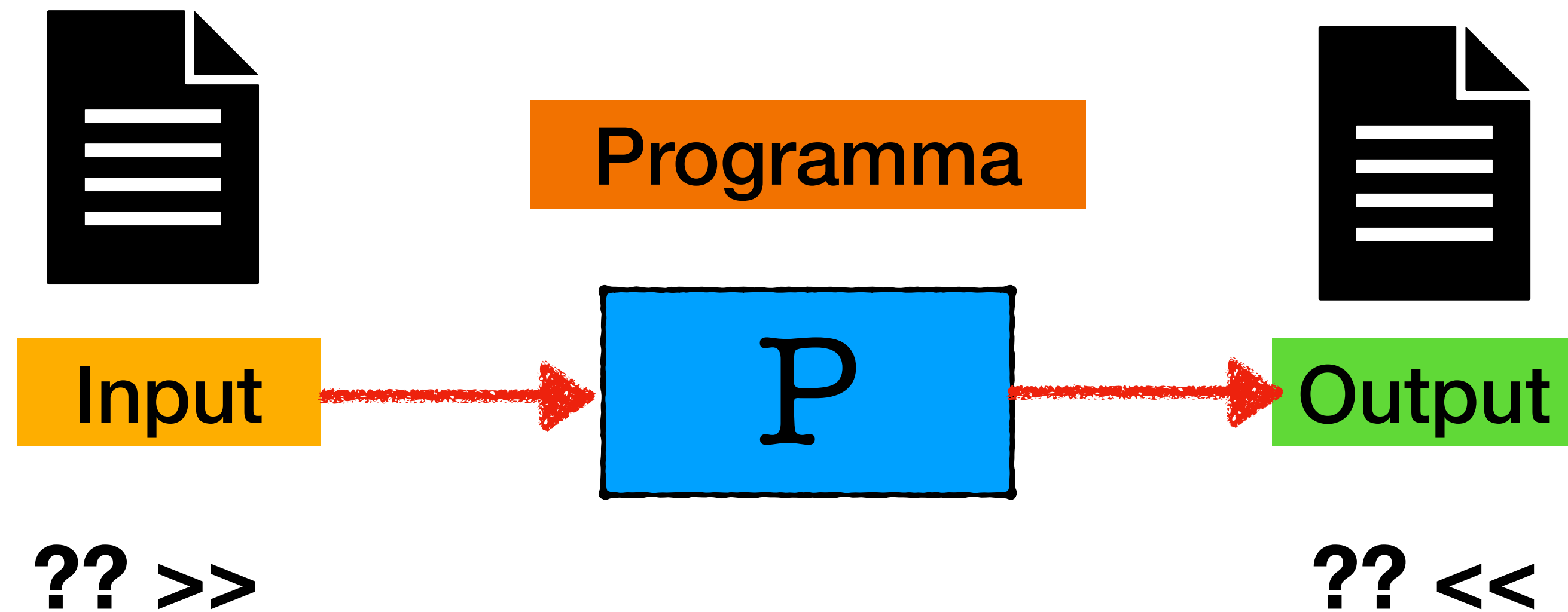
/home/tama/Desktop/dati.dat

```
'N' 'e' 'l' ' ' 'm' 'e' 'z' 'z' 'o' 'd' 'e' 'l' ' ' 'c'  
'a' 'm' 'm' 'i' 'n' ' ' 'd' 'i' ' ' 'n' 'o' 's' 't' 'r' 'a'  
' ' 'v' 'i' 't' 'a' '\n' 'm' 'i' ' ' 'r' 'i' 't' 'r' 'o' 'v'  
'a' 'i'
```

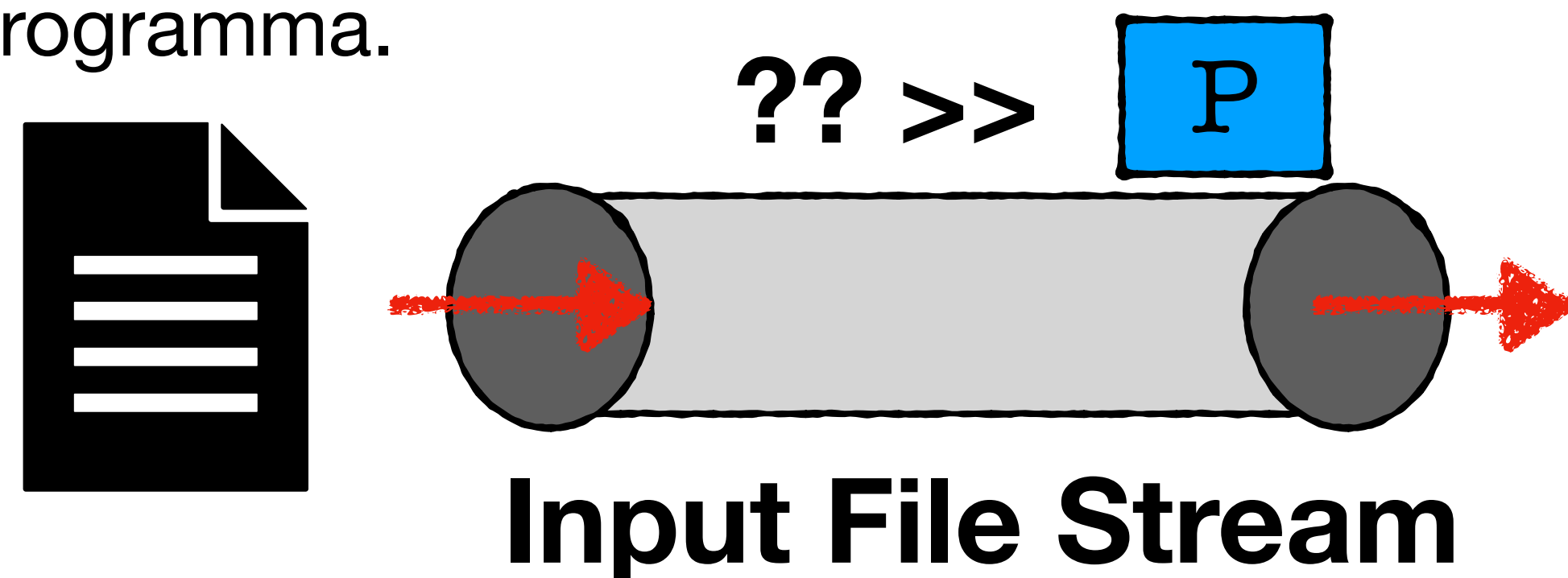
....

Attraverso opportune invocazioni dei servizi del sistema operativo, che gestisce l'accesso al disco fisso, il contenuto dei file può essere reso accessibile ai programmi.

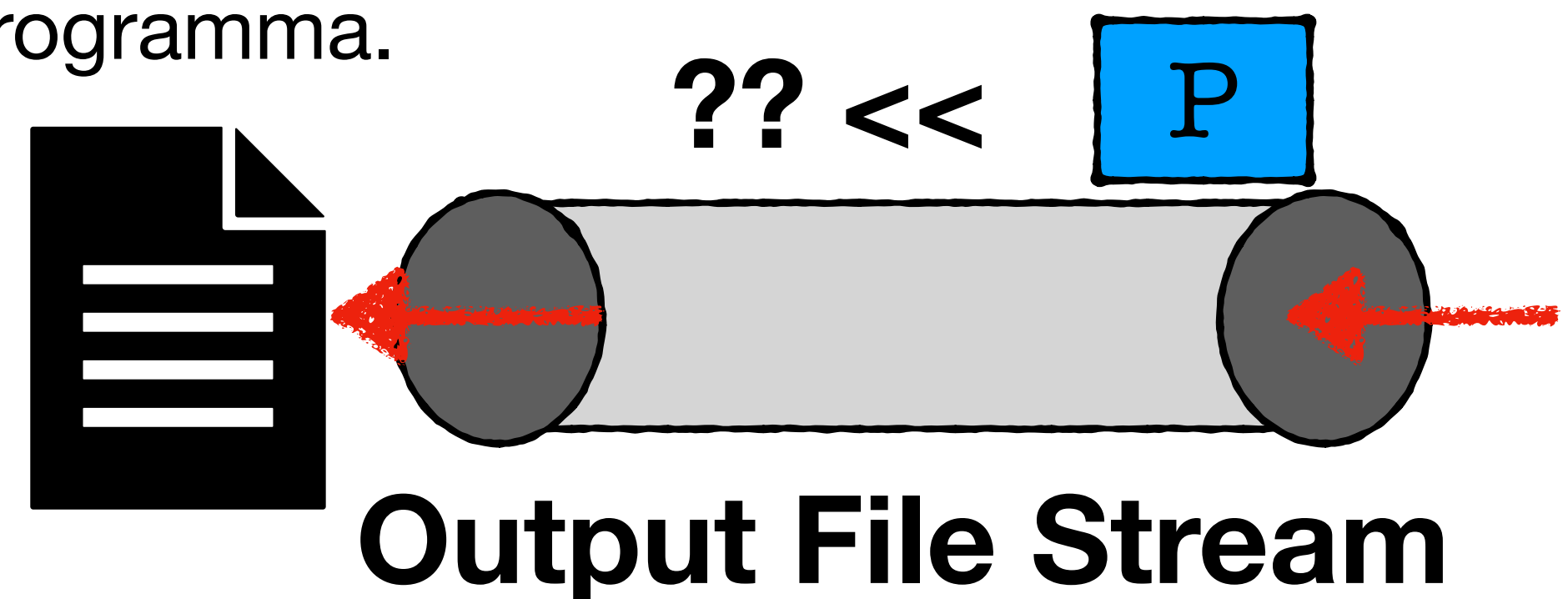
# Flussi di dati da e verso files



- Un file può essere la SORGENTE di un flusso di informazioni, che viaggiano verso il programma.

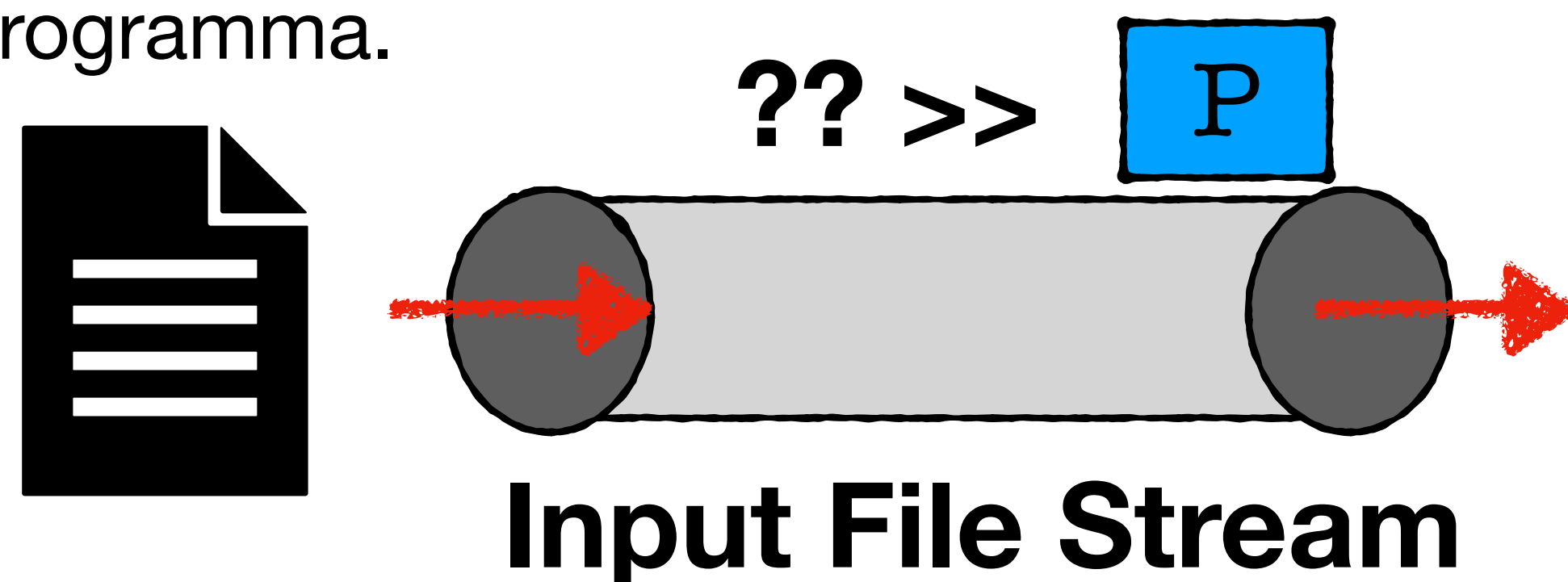


- Un file può essere la DESTINAZIONE di un flusso di informazioni, che originano dal programma.

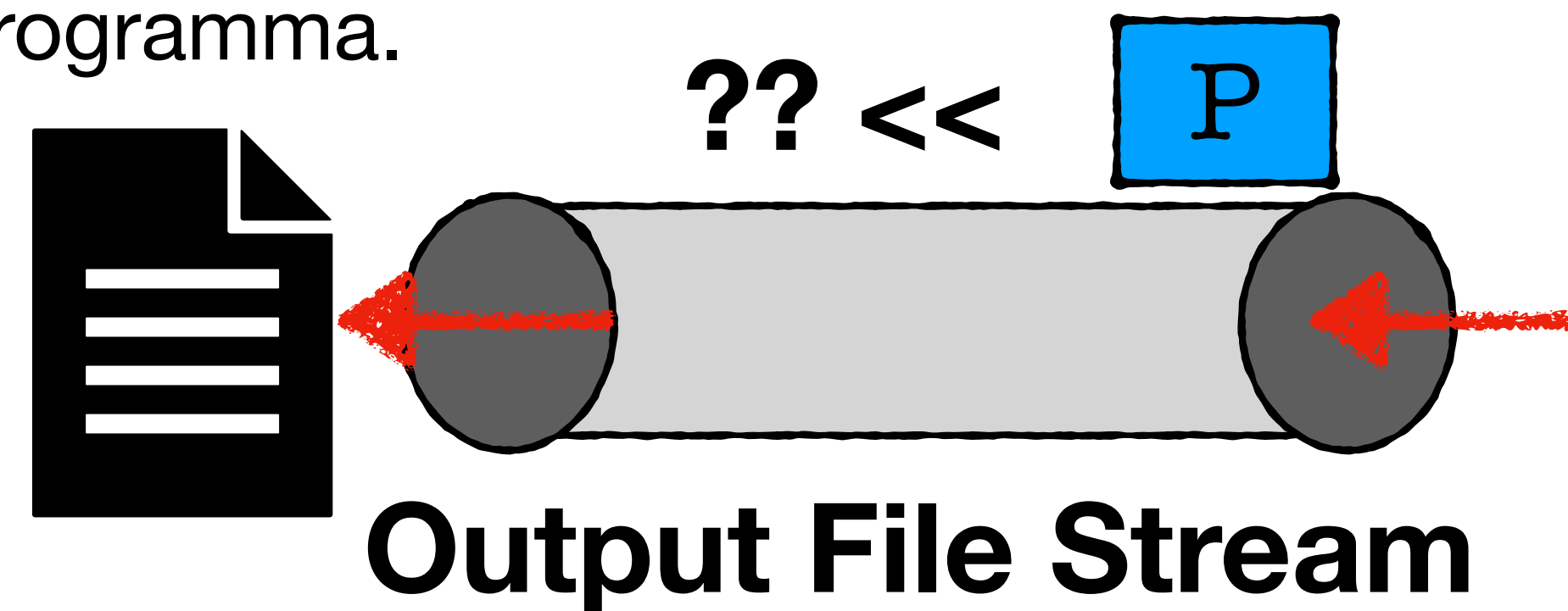


# Che cosa cambia rispetto a std in/out?

- Un file può essere la SORGENTE di un flusso di informazioni, che viaggiano verso il programma.



- Un file può essere la DESTINAZIONE di un flusso di informazioni, che originano dal programma.



- Tastiera e monitor sono unici e sono chiaramente dispositivi di input e output rispettivamente.
- I files sono tanti, e un file può essere sia letto (quindi essere una SORGENTE di dati) che scritto (e quindi essere una DESTINAZIONE).

- È necessario esplicitare l'operazione di associazione di un file ad un flusso
- ...e la direzione del flusso



# Files: cassetta degli attrezzi

**#include <fstream>**

## Libreria fstream

- Include una serie di strumenti/entità/classi per la gestione dell'I/O da/verso files.
- Associazione file in lettura/scrittura a stream
- Controllo dello stato dello stream (errore/stream esaurito)
- Rilascio dell'associazione di un file ad un flusso.

## Attenzione

- La libreria fstream mette a disposizione i suddetti strumenti sotto forma di classi, ovvero particolari "tipi di dato" capaci di fare anche delle azioni, ovvero mettere a disposizione dei servizi.
- Non deve stupire: il C++ è un linguaggio ad oggetti. Ma noi non tratteremo l'argomento per ora...
- Adotteremo invece un approccio "ignorante", imparando i comandi necessari ad una gestione minimale degli stream da/verso files e la loro semantica.





# Associazione file a input stream

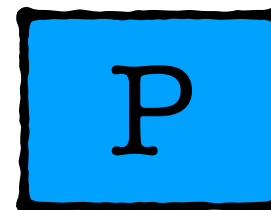
dati.dat



```
ifstream flusso_in;
```

```
flusso_in.open("dati.dat");
```

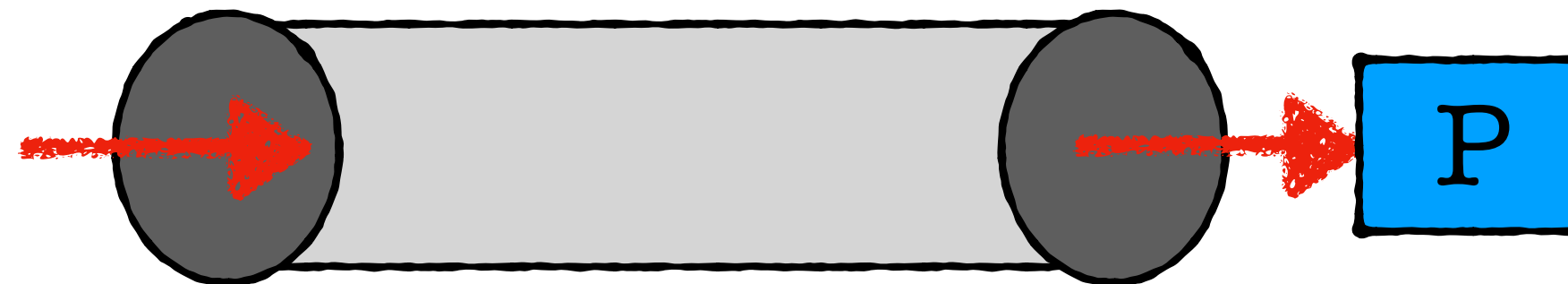
?? >>



- La variabile di "tipo" stream di input viene creata.
- Lo stream di input viene associato, se tutto va bene (il file esiste) al file "dati.dat".
- Lo stream di input si usa in modo "normale"



dati.dat



## Input File Stream

# Uso di file input stream

```
ifstream flusso_in;
```

```
int a;  
float b;  
char c;
```

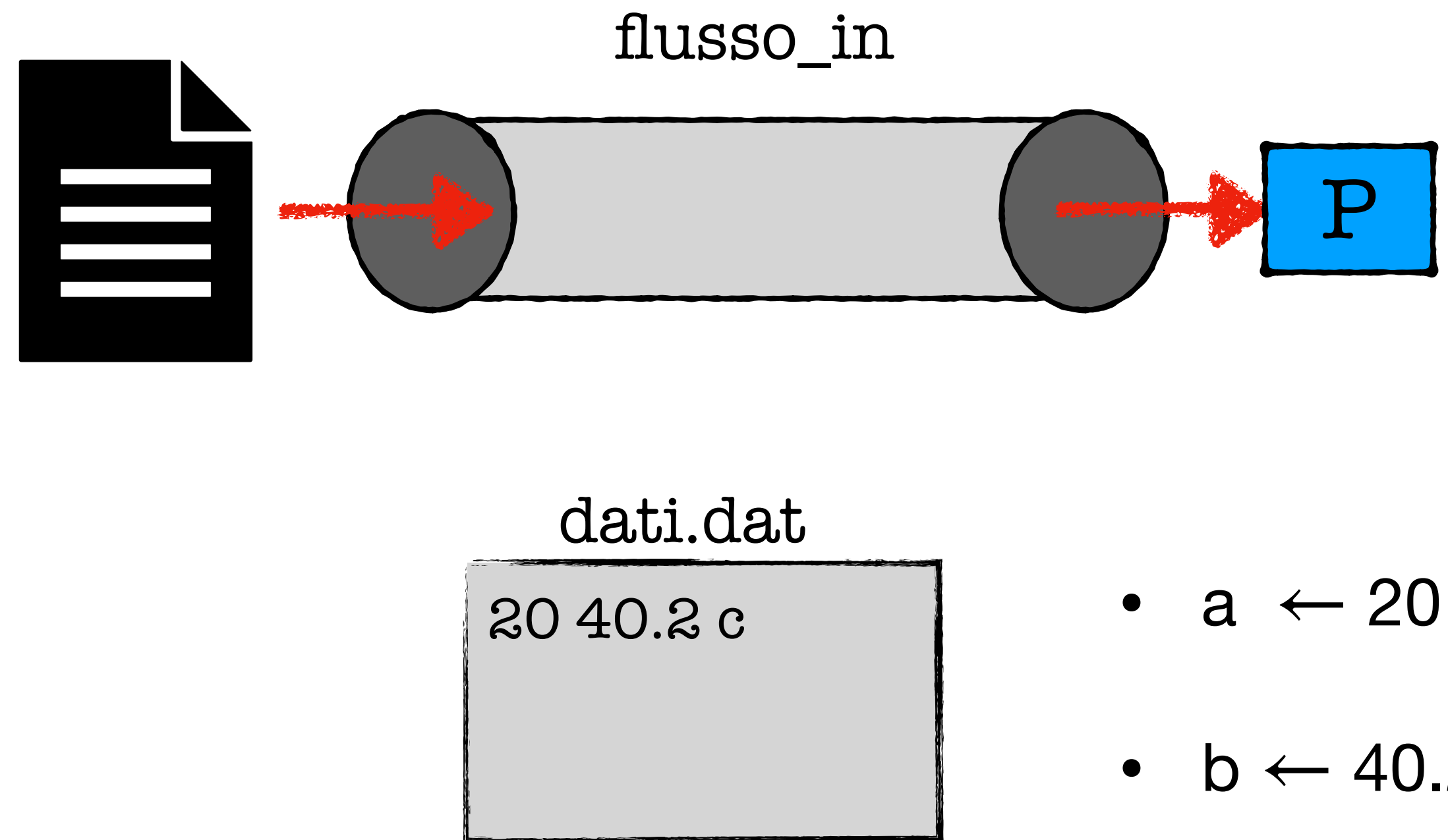
```
flusso_in.open("dati.dat");
```

```
flusso_in >> a >> b >> c;
```

Finito di usare lo stream, rilascio il file.

```
flusso_in.close();
```

Rilasciato il file, lo stream di input `flusso_in` può, nel caso essere associato nuovamente ad un file (anche lo stesso)



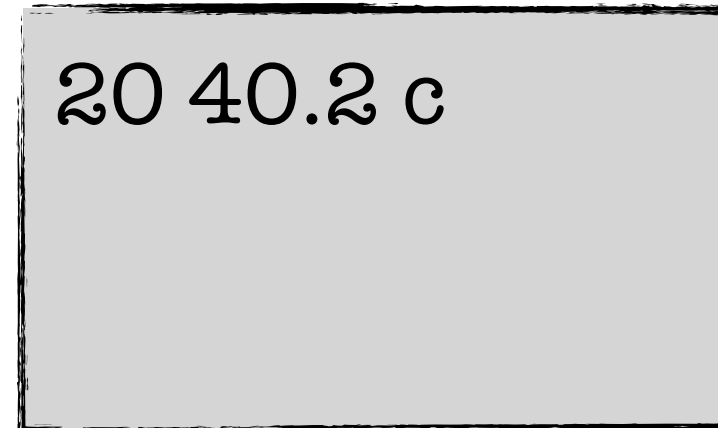
# Uso flussi da file....

- **Come per il cin** i caratteri inseriti nel flusso di input vengono consumati "token a token" dall'operatore di estrazione (>>).

```
int a;  
float b;  
char c;
```

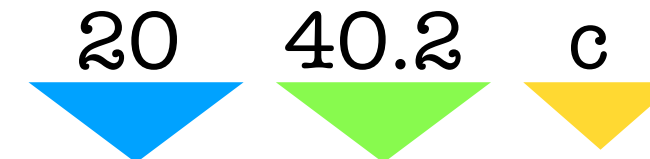
```
cin >> a >> b >> c;
```

dati.dat



20 40.2 c

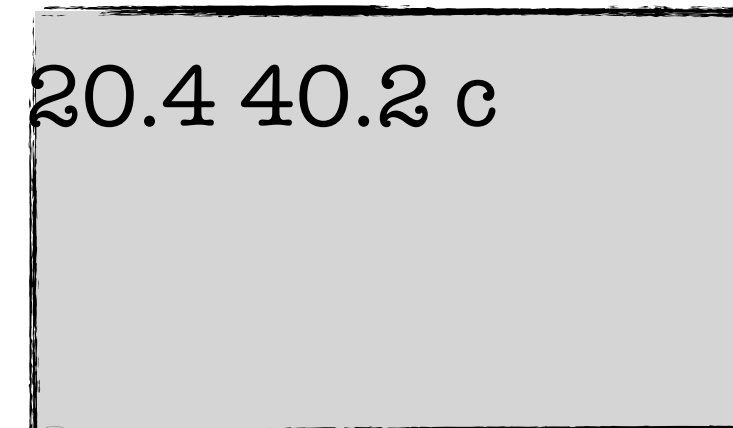
20 40.2 c



- $a \leftarrow 20$
- $b \leftarrow 40.2$
- $c \leftarrow 'c'$

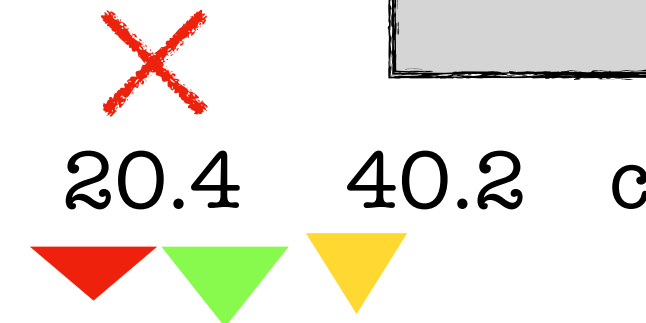
- la conformazione dei "token" dipende dal tipo della variabile di destinazione.
- ~~I dati inseriti da tastiera vengono inseriti nel flusso quando viene inserito il carattere "invio/return".~~
- Se un token è mal formato, lo stream fa cose apparentemente strane.....

dati\_1.dat



20.4 40.2 c

20.4 40.2 c



- $a \leftarrow 20$
- $b \leftarrow 0.4$
- $c \leftarrow '4'$

# file input stream: dettagli

- Un file può essere usato in lettura se sappiamo la natura dei dati in esso contenuti, ovvero abbiamo concordato con chi fornisce il file il **FORMATO**.

```
cin >> a >> b >> c;
```

dati.dat

20 40.2 c
-----------

- $a \leftarrow 20$
- $b \leftarrow 40.2$
- $c \leftarrow 'c'$

- Devo sapere che il primo valore è un intero, il secondo un numero razionale e il terzo un carattere.
- Se un file **non** rispetta il formato concordato, l'effetto è quello di avere token mal formati, con le conseguenze del caso.....
- "il file misure.dat contiene, su ciascuna riga, un intero, un razionale e un carattere..."

✗

20.4 40.2 c

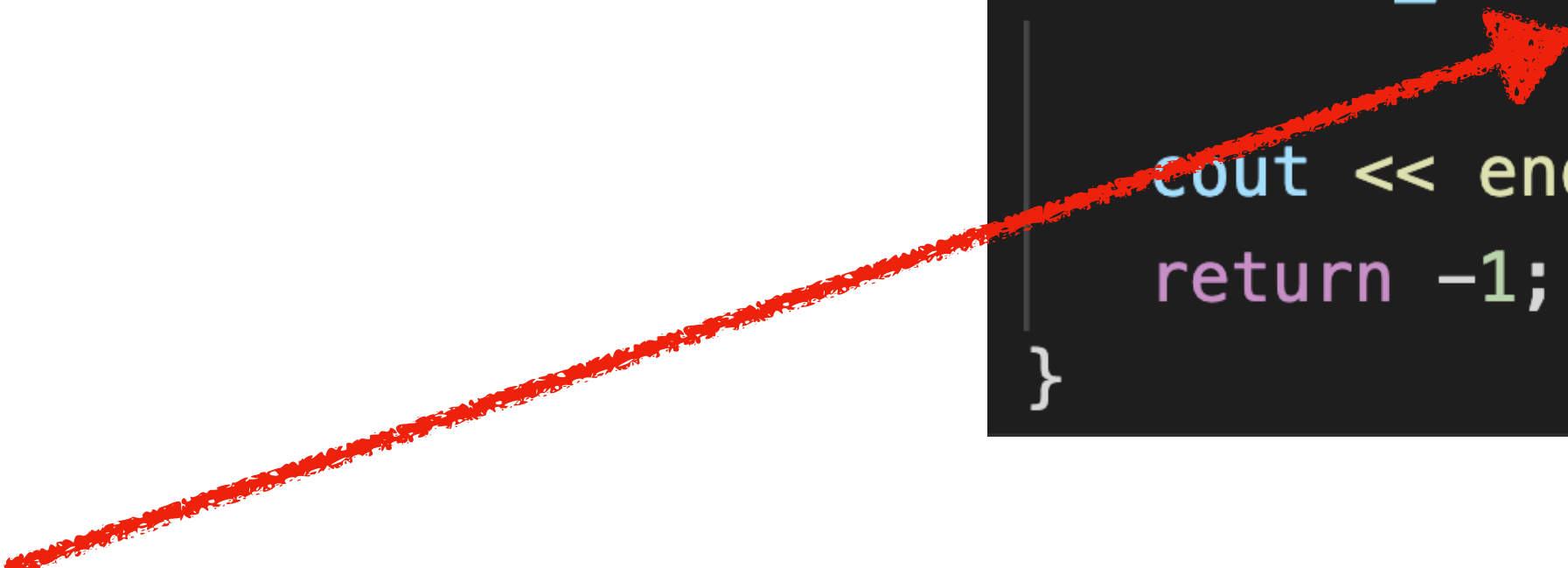
▼ ▼ ▼

- $a \leftarrow 20$
- $b \leftarrow 0.4$
- $c \leftarrow '4'$

# file input stream: errori

- **Errori:**
  - Se il file associato allo stream di ingresso non esiste, lo stream si "rompe"...
  - ...e lo stream fa cose apparentemente strane.....
  - Possiamo controllare lo stato dello stream...

```
if(flusso_in.fail()){  
    cout << endl << "Problema apertura file" << endl;  
    return -1;  
}
```



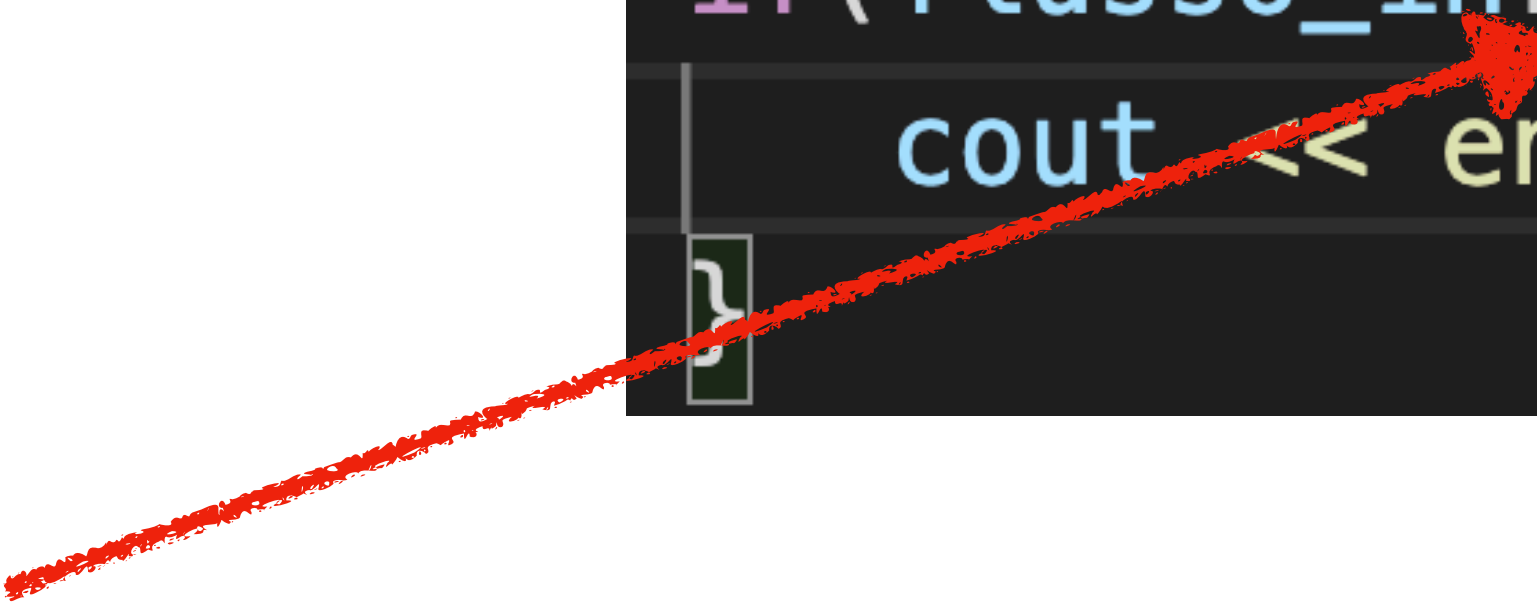
Chiediamo se lo stream è in "fail". Se è rotto la risposta è **vero**

Possiamo gestire lo stato di fail in diversi modi.  
Qui usciamo dal programma!

# file input stream: fine file

- La lettura dei dati da file fa avanzare un cursore (testina) che indica il prossimo carattere da leggere nel file.
- In questo senso il file viene "consumato", anche se il contenuto rimane invariato.
- Quando il cursore (testina) raggiunge la fine del file, lo stream entra in uno stato di End Of File (EOF)

```
if(flusso_in.eof()){  
    cout << endl << "File finito" << endl;  
}
```



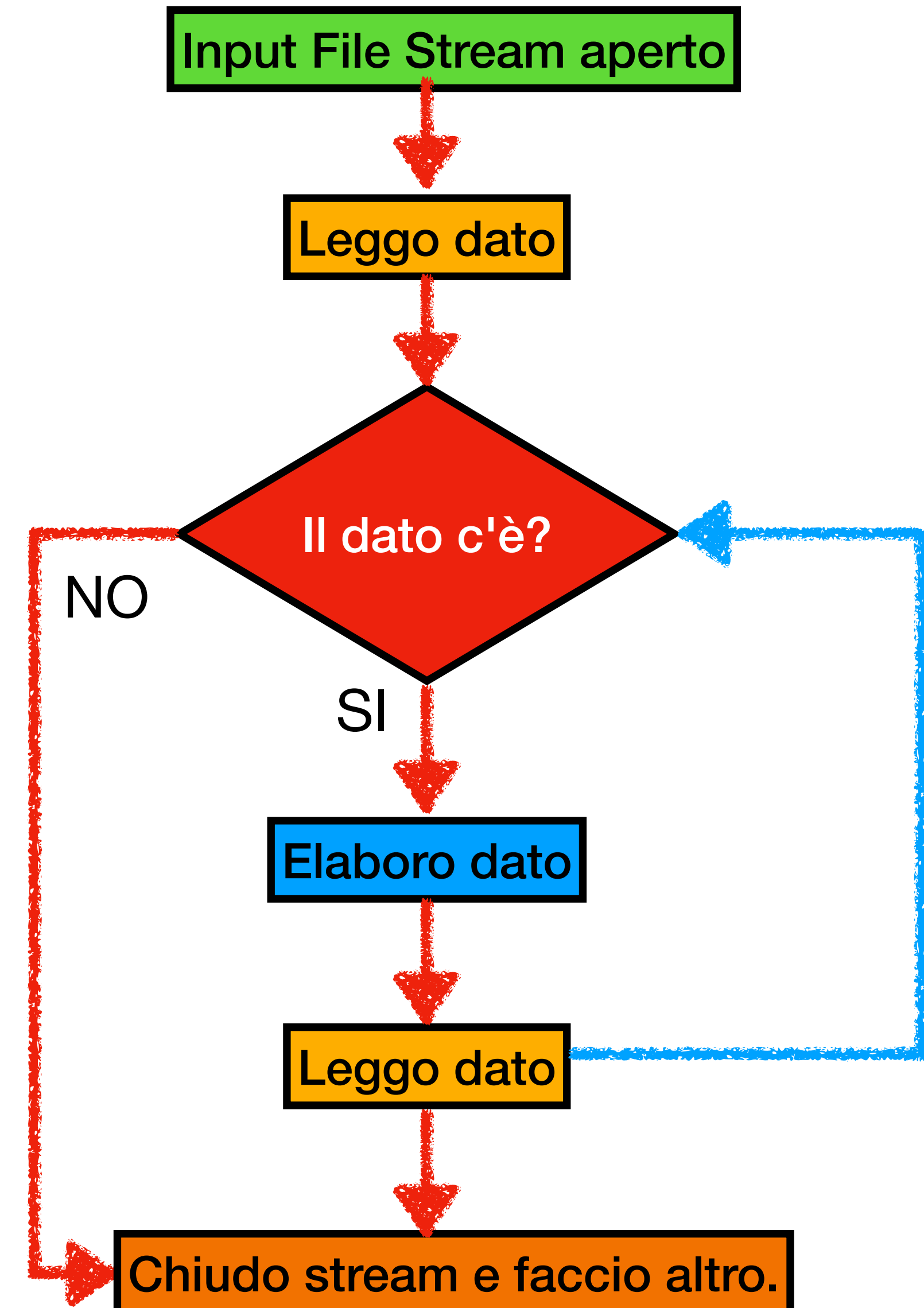
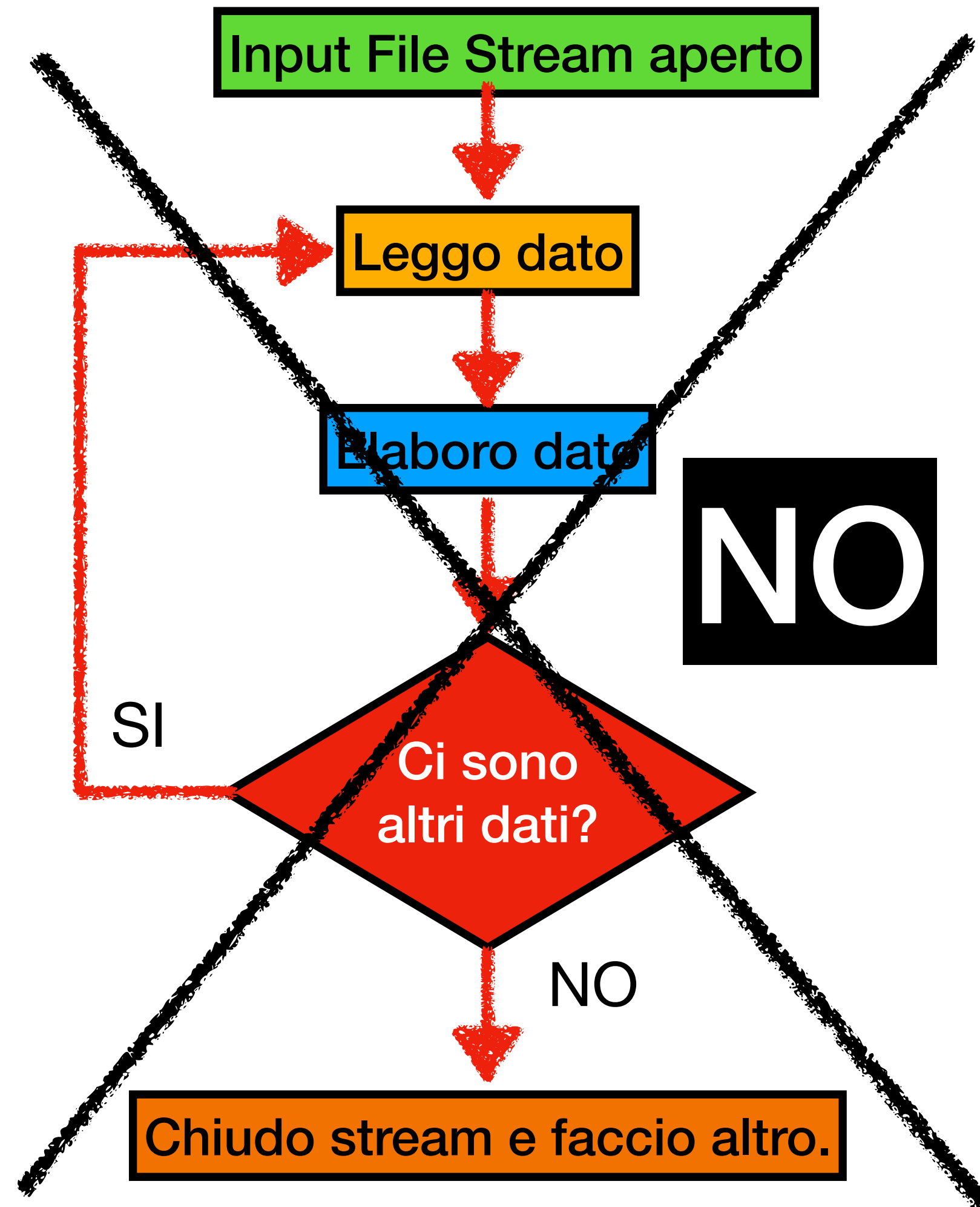
Chiediamo se lo stream è in "EOF". Se il cursore ha raggiunto la fine del file, risposta è **vero**...

...e potremo agire di conseguenza....



# Il ciclo Spoletini

## Come si leggono i dati da file





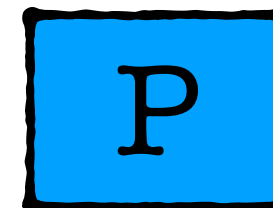
# Associazione file a output stream

risultati.dat



```
ofstream flusso_out;
```

?? <<

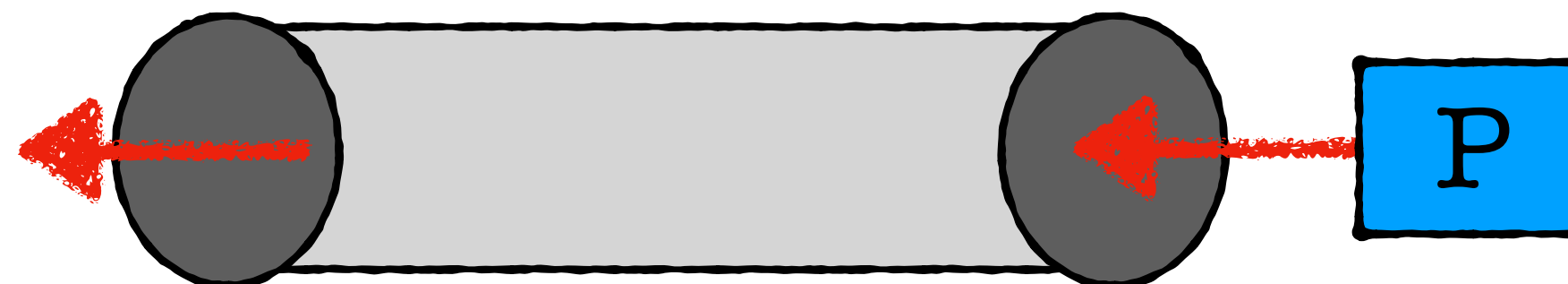


```
flusso_out.open("risultati.dat");
```

- La variabile di "tipo" stream di output viene creata.
- Lo stream di output viene associato al file "risultati.dat".
  - Se "risultati.dat" **esiste**, viene **pulito** e **sovrascritto**.
  - Se "risultati.dat" **non esiste**, viene **creato**.



risultati.dat



- Lo stream di input si usa in modo "normale"

## Output File Stream

# Uso di file output stream

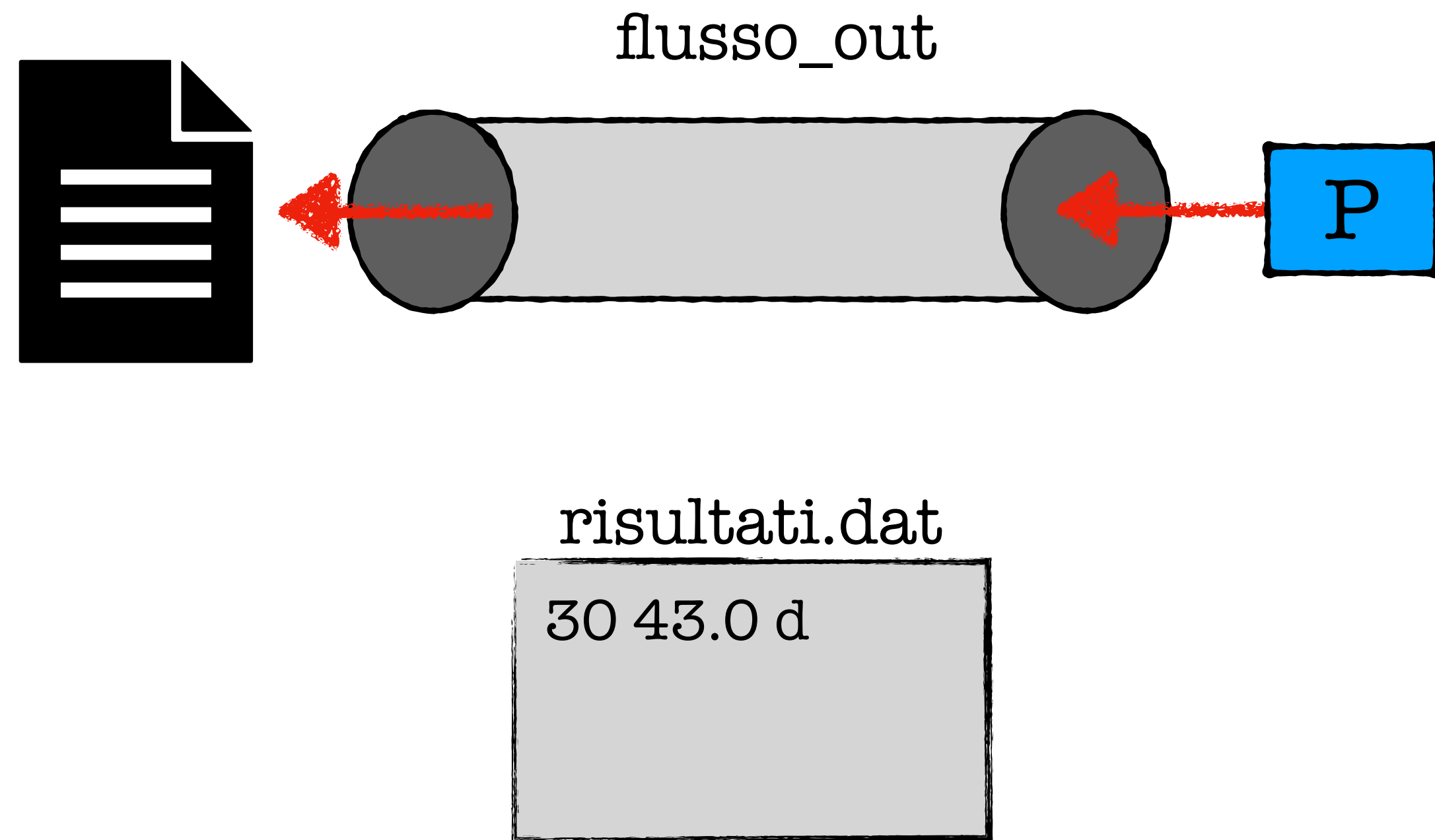
```
ofstream flusso_out;
```

```
flusso_out.open("risultati.dat");
```

```
a = 30;  
b = 43.0;  
c = 'd';
```

```
flusso_out << a << " " << b << " " << c;
```

```
flusso_out.close();
```



Finito di usare lo stream, **rilascio** il file.

Rilasciato il file, lo stream di output `flusso_out` può, nel caso essere associato nuovamente ad un file....

# file output stream: dettagli

- Il comportamento è molto simile a quello del cout.
- Più difficile fare errori: se il file su cui scrivere non esiste viene creato
- Ma attenzione a non cancellare il contenuto di files accidentalmente!